The Embedded I/O Company

TEWS TECHNOLOGIES

TIP866-SW-42

VxWorks Device Driver

8 Channel Serial Interface IP

Version 2.1.x

User Manual

Issue 2.1.0

April 2013

TEWS TECHNOLOGIES GmbH
Am Bahnhof 7 25469 Halstenbek, Germany
Phone: +49 (0) 4101 4058 0 Fax: +49 (0) 4101 4058 19
e-mail: info@tews.com www.tews.com

TIP866-SW-42

VxWorks Device Driver

8 Channel Serial Interface IP

Supported Modules:
TIP866

This document contains information, which is proprietary to TEWS TECHNOLOGIES GmbH. Any reproduction without written permission is forbidden.

TEWS TECHNOLOGIES GmbH has made any effort to ensure that this manual is accurate and complete. However TEWS TECHNOLOGIES GmbH reserves the right to change the product described in this document at any time without notice.

TEWS TECHNOLOGIES GmbH is not liable for any damage arising out of the application or use of the device described herein.

©2005-2013 by TEWS TECHNOLOGIES GmbH

Issue	Description	Date
1.0	First Issue	August 1996
1.1	New versions (baud rates up to 460800 baud)	June 1997
1.2	New control commands	August 1997
1.3	Advanced configuration description	April 1998
1.4	New *ioctl()* function	February 1999
1.5	General Revision	November 2003
1.8.0	General revision, IPAC CARRIER driver support and new user interface symbols	November 24, 2005
1.9.0	New error code for tip866DevCreate(), New address TEWS LLC	August 2, 2007
1.9.1	Carrier Driver description added	June 24, 2008
2.0.0	Function of tdrv002Drv() and tdrv002DevCreate() modified, SMP-Support added , Address TEWS LLC removed	January 19, 2011
2.1.0	Prototypes for 64-bit support modified Chapter “Known Issue with Interrupt Execution Timer” added	April 17, 2013

Table of Contents

1 INTRODUCTION 4
1.1 Device Driver 4
1.2 IPAC Carrier Driver 5
2 INSTALLATION 6
2.1 Include Device Driver in VxWorks Projects 6
2.2 System Resource Requirement 7
2.3 Known Issue with Interrupt Execution Time 7
3 I/O SYSTEM FUNCTIONS 8
3.1 tip866Drv 8
3.2 tip866DevCreate 10
4 I/O FUNCTIONS 13
4.1 open 13
4.2 close 15
4.3 read 17
4.4 write 19
4.5 ioctl 21
4.5.1 FIOBAUDRATE 23
4.5.2 FIO_TIP866_FIFO 25
4.5.3 FIO_TIP866_DATABITS 27
4.5.4 FIO_TIP866_STOPBITS 28
4.5.5 FIO_TIP866_PARITY 29
4.5.6 FIO_TIP866_ENABLEHWHS 30
4.5.7 FIO_TIP866_DISABLEHWHS 31
4.5.8 FIO_TIP866_CHECKBREAK 32
4.5.9 FIO_TIP866_SETBREAK 33
4.5.10 FIO_TIP866_CLEARBREAK 34
4.5.11 FIO_TIP866_RECONFIGURE 35
4.5.12 FIO_TIP866_CHECKERRORS 36

1 Introduction

1.1 Device Driver

The TIP866-SW-42 VxWorks device driver software allows the operation of the TIP866 IP conforming to the VxWorks I/O system specification. This includes a device-independent basic I/O interface with open(), close(), read(), write() and ioctl() functions and a buffered I/O interface (fopen(), fprintf(), fscanf(), …).

The TIP866-SW-42 driver includes the following functions supported by the VxWorks tty driver support library:

- ring buffering of input and output
- raw mode
- optional line mode with backspace and line-delete functions
- optional processing of X-on/X-off
- optional RETURN/LINEFEED conversion
- optional echoing of input characters
- optional stripping of the parity bit from 8 bit input
- option special characters for shell abort and system restart

Additional the following functions are supported:

- select receive and transmit FIFO trigger levels
- use 5..8 bit data words
- use 1, 1.5 or 2 stop bits
- optional even or odd parity
- enable/disable hardware handshake (only in FIFO mode)

The TIP866-SW-42 supports the modules listed below:

TIP866-10	8 Channel Serial (RS232)	IndustryPack
TIP866-11	8 Channel Serial (TTL)	IndustryPack
TIP866-20	8 Channel Serial (RS422)	IndustryPack

To get more information about the features and use of the supported devices it is recommended to read the manuals listed below.

TIP866 User Manual
TIP866 Engineering Manual (16C654 UART controller information included)
CARRIER-SW-42 IPAC Carrier User Manual

1.2 IPAC Carrier Driver

IndustryPack (IPAC) carrier boards have different implementations of the system to IndustryPack bus bridge logic, different implementations of interrupt and error handling and so on. Also the different byte ordering (big-endian versus little-endian) of CPU boards will cause problems on accessing the IndustryPack I/O and memory spaces.

To simplify the implementation of IPAC device drivers which work with any supported carrier board, TEWS TECHNOLOGIES has designed a so-called Carrier Driver that hides all differences of different carrier boards under a well-defined interface.

The TEWS TECHNOLOGIES IPAC Carrier Driver CARRIER-SW-42 is part of this TIP866-SW-42 distribution. It is located in directory CARRIER-SW-42 on the corresponding distribution media.

This IPAC Device Driver requires a properly installed IPAC Carrier Driver. Due to the design of the Carrier Driver, it is sufficient to install the IPAC Carrier Driver once, even if multiple IPAC Device Drivers are used.

Please refer to the CARRIER-SW-42 User Manual for a detailed description how to install and setup the CARRIER-SW-42 device driver, and for a description of the TEWS TECHNOLOGIES IPAC Carrier Driver concept.

2 Installation

The following files and directories are located on the distribution media:

Directory path 'TIP866-SW-42':

tip866drv.c	TIP866 device driver source
tip866def.h	TIP866 driver include file
tip866.h	TIP866 include file for driver and application
tip866conf.h	TIP866 driver configuration file
tip866exa.c	Example application
include/ipac_carrier.h	Carrier driver interface definitions
TIP866-SW-42-2.1.0.pdf	PDF copy of this manual
Release.txt	Release information
ChangeLog.txt	Release history

2.1 Include Device Driver in VxWorks Projects

For including the TIP866-SW-42 device driver into a VxWorks project (e.g. Tornado IDE or Workbench) follow the steps below:

(1) Copy the files from the distribution media into a subdirectory in your project path. (For example: ./TIP866)

(2) Add the device drivers C-files to your project.

(3) Now the driver is included in the project and will be built with the project.

For a more detailed description of the project facility please refer to your VxWorks User's Guide (e.g. Tornado, Workbench, etc.)

2.2 System Resource Requirement

The table gives an overview over the system resources that will be needed by the driver.

Resource	Driver requirement	Devices requirement
Memory	< 1 KB	< 1 KB
Stack	< 1 KB	---
Semaphores	1	---

Memory and Stack usage may differ from system to system, depending on the used compiler and its setup.

The following formula shows the way to calculate the common requirements of the driver and devices.

*<total requirement> = <driver requirement> + (<number of devices> * <device requirement>)*

The maximum usage of some resources is limited by adjustable parameters. If the application and driver exceed these limits, increase the according values in your project.

The VxWorks terminal manager will require additional resources per installed channel for management information and FIFO buffers.

2.3 Known Issue with Interrupt Execution Time

In newer systems (VxWorks 6.x) a long interrupt execution time may lead to a work queue overflow, which may result in system crash or error state. If such a situation occurs while a data transfer is in progress there are two ways to solve the problem: first the FIFO-trigger-levels can be adapted to decrease the interrupt execution time, and secondly the Work Queue Size can be increased (value of WIND_JOBS_MAX). Please refer to the VxWorks documentation for description of project configuration.

3 I/O System Functions

This chapter describes the driver-level interface to the I/O system. The purpose of these functions is to install the driver in the I/O system, add and initialize devices.

3.1 tip866Drv

NAME

tip866Drv() - installs the TIP866 driver in the I/O system

SYNOPSIS

#include “tip866.h”

```
STATUS tip866Drv
(
      void
)
```

DESCRIPTION

This function initializes the TIP866 driver and installs it in the I/O system. The driver searches for TIP866 modules and prepares structures for device creation.

The call of this function is the first thing the user has to do before adding any device to the system or performing any I/O request.

EXAMPLE

```
#include "tip866.h”

STATUS    result;

/*------ Initialize Driver -----*/
result = tip866Drv();
if (result == ERROR)
{
     /* Error handling */
}
```

RETURNS

OK, or ERROR if the function fails.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

VxWorks Programmer's Guide: I/O System

3.2 tip866DevCreate

NAME

tip866DevCreate() – Add a TIP866 serial channel device to the VxWorks system

SYNOPSIS

#include “tip866.h”

```
STATUS tip866DevCreate
(
        char        *name,
        int         devIdx,
        int         funcType,
        void        *pParam
)
```

DESCRIPTION

This routine adds the selected device to the VxWorks system. The device hardware will be setup and prepared for use.

This function must be called before performing any I/O request to this device.

PARAMETER

name

This string specifies the name of the device that will be used to identify the device, for example for *open()* calls.

devIdx

This index number specifies the TIP866 serial channel device to add to the system.

If modules of the same type are installed the channel numbers will be assigned in the order the IPAC CARRIER *ipFindDevice()* function will find the devices.

Example: A system with one TIP866-10 on 1st slot and two TIP866-11 on 2nd and 3rd slot will assign the following device indexes:

Module	Device Index
TIP866-10 (Channel 1..8)	0..7
TIP866-11 (Channel 1..8)	8..15
TIP866-11 (Channel 1..8)	16..23

funcType

This parameter is unused and should be set to *0*.

pParam

This parameter points to a structure (*TIP866_DEVCONFIG*) containing the default configuration of the channel.

```
typedef struct
{
    int rdBufSize;
    int wrtBufSize;
} TIP866_DEVCONFIG;
```

rdBufSize

Size of input ring buffer in bytes

wrtBufSize

Size of output ring buffer in bytes

EXAMPLE

```
#include "tip866.h"

STATUS             result;
TIP866_DEVCONFIG   tip866Conf;

/*-----------------------------------------------
  Create the device "/tip866/0" for the first serial
  channel of the first found module
  -----------------------------------------------*/
tip866Conf.rdBufSize =  512;
tip866Conf.wrtBufSize = 512;

result = tip866DevCreate(    "/tip866/0",
                             0,
                             0,
                             (void*)&tip866Conf);
if (result == OK)
{
     /* Device successfully created */
}
else
{
     /* Error occurred when creating the device */
}
```

RETURNS

OK, or ERROR. If the function fails, an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

Error Code	Description
S_ioLib_NO_DRIVER	Driver not installed, run tip866Drv()
S_tip866Drv_IARG	Invalid argument in device configuration buffer. Please check all arguments given to tip866DevCreate().
S_tip866Drv_EXISTS	The specified Device has been created already

SEE ALSO

VxWorks Programmer's Guide: I/O System

4 I/O Functions

4.1 open

NAME

open() - open a device or file.

SYNOPSIS

```
int open
(
        const char  *name,
        int         flags,
        int         mode
)
```

DESCRIPTION

Before I/O can be performed to the TIP866 device, a file descriptor must be opened by invoking the basic I/O function *open().*

PARAMETER

name

Specifies the device which shall be opened. The name specified in tip866DevCreate() must be used.

flags

Not used

mode

Not used

EXAMPLE

```
int      fd;

/*----------------------------------------
  Open the device named "/tip866/0" for I/O
  ----------------------------------------*/
fd = open("/tip866/0", 0, 0);
if (fd == ERROR)
{
     /* Handle error */
}
```

RETURNS

A device descriptor number, or ERROR. If the function fails, an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - *open()*

4.2 close

NAME

close() – close a device or file

SYNOPSIS

```
STATUS close
(
      int           fd
)
```

DESCRIPTION

This function closes opened devices.

PARAMETER

fd

This file descriptor specifies the device to be closed. The file descriptor has been returned by the *open()* function.

EXAMPLE

```
int       fd;
STATUS    result;

/*----------------
  close the device
  ----------------*/
result = close(fd);
if (result == ERROR)
{
     /* Handle error */
}
```

RETURNS

OK or ERROR. If the function fails, an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - close()

4.3 read

NAME

read() – read data from a specified device.

SYNOPSIS

```
int read
(
        int             fd,
        char            *buffer,
        size_t          maxbytes
)
```

DESCRIPTION

This function can be used to read data from the device.

PARAMETER

fd

This file descriptor specifies the device to be used. The file descriptor has been returned by the *open()* function.

buffer

This argument points to a user supplied buffer. The returned data will be filled into this buffer.

maxbytes

This parameter specifies the maximum number of read bytes (buffer size).

EXAMPLE

```
#define   BUFSIZE   1024

int             fd;
char            buffer[BUFSIZE];
int             retval;

…
```

…

```
/*-------------------------------------------------------
  Read data from the TIP866 serial channel connected to “fd”
  -------------------------------------------------------*/
retval = read(fd, buffer, BUFSIZE);
if (retval != ERROR)
{
     printf(“%d bytes read\n”, retval);
}
else
{
     /* handle the read error */
}
```

RETURNS

Number of bytes read or ERROR. If the function fails, an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, tyRead, basic I/O routine - read()

4.4 write

NAME

write() – write data from a buffer to a specified device.

SYNOPSIS

```
int write
(
        int             fd,
        char            *buffer,
        size_t          nbytes
)
```

DESCRIPTION

This routine writes *nbytes* bytes from *buffer* to the specified serial channel connected with the device descriptor *fd*.

PARAMETER

fd

This file descriptor specifies the device to be used. The file descriptor has been returned by the *open()* function.

buffer

This argument points to a user supplied buffer. The data of the buffer will be written to the device.

nbytes

This parameter specifies the number of bytes to write.

EXAMPLE

```
int             fd;
char            buffer[] = “Hello World\n”;
int             retval;

…
```

…

```
/*-----------------------------------------------------
  Write data to a TIP866 serial channel connected to fd
  -----------------------------------------------------*/
retval = write(fd, buffer, strlen(buffer));
if (retval != ERROR)
{
     printf("%d bytes written\n", retval);
}
else
{
     /* handle the write error */
}
```

RETURNS

Number of bytes written or ERROR. If the function fails, an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, tyWrite, basic I/O routine - write()

4.5 ioctl

NAME

ioctl() - performs an I/O control function.

SYNOPSIS

#include “tip866.h”

```
int ioctl
(
        int                     fd,
        int                     request,
        TIP866_IOCTL_ARG_T      arg
)
```

DESCRIPTION

Special I/O operation that do not fit to the standard basic I/O calls will be performed by calling the *ioctl* function with a specific function code and an optional function dependent argument.

The TIP866 device driver uses the standard tty driver support library tyLib. For details of supported *ioctl* functions see VxWorks Reference Manual: tyLib and VxWorks Programmer’s Guide: I/O system.

PARAMETER

fd

This file descriptor specifies the device to be used. The file descriptor has been returned by the *open()* function.

request

This argument specifies the function that shall be executed.
Following functions are defined:

Function	Description
FIOBAUDRATE	Set baud rate
FIO_TIP866_FIFO	Set receive and transmit FIFO trigger levels
FIO_TIP866_DATABITS	Set data word length
FIO_TIP866_STOPBITS	Set number of stop bits
FIO_TIP866_PARITY	Set parity checking
FIO_TIP866_ENABLEHWHS	Enable hardware handshake
FIO_TIP866_DISABLEHWHS	Disable hardware handshake
FIO_TIP866_CHECKBREAK	Check for pending break
FIO_TIP866_SETBREAK	Set break condition
FIO_TIP866_CLEARBREAK	Clear break condition
FIO_TIP866_RECONFIGURE	Restart channel
FIO_TIP866_CHECKERRORS	Read error status

arg

This parameter depends on the selected function (request). How to use this parameter is described below with the function.

RETURNS

Function dependent value (described with the function) or ERROR. If the function fails, an error code will be stored in *errno*.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

SEE ALSO

ioLib, basic I/O routine - ioctl()

4.5.1 FIOBAUDRATE

This I/O control function sets the baud rate of the certain serial channel. The function specific control parameter **arg** specifies the new baud rate.

The FIOBAUDRATE function is a standard function, but there are a few points to pay attention to. The selected baud rate is always set to the next selectable value.

arg

This parameter sets the new baud rate. For TIP866-10 the maximum baud rate is 115200 Baud, for TIP866-11 and TIP866-20 the maximum baud rate is 460800 Baud.

To get a list of valid baud rates use the following formula:
For baud rates less than or equal to 50 Baud: baud rate = 115200 / n
For baud rates higher than 50 Baud: baud rate = 460800 / n
For both equations the divisor n should be in range of 1 to 65536

Desired baud rate	Selected baud rate
9600	9600
9500	9600
100000	115200
115200	115200

Higher baud rates shall be used with enabled FIFO to avoid losing data.

EXAMPLE

```
#include “tip866.h”

int           fd;
int           retval;

/*---------------------------
  Set baud rate to 57600 Baud
  ---------------------------*/
retval = ioctl(fd, FIOBAUDRATE, 57600);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual) or a driver set code described below. Function specific error codes will be described with the function.

Error Code	Description
S_tip866Drv_IARG	Invalid or unsupported baud rate.

4.5.2 FIO_TIP866_FIFO

This I/O control function sets the receive and transmit FIFO trigger levels. The function specific control parameter **arg** specifies the new configuration.

arg

The special argument arg is split into four bytes:
xxxxTTRR
xxxx the two most significant bytes are unused

TT the third byte selects the transmitter trigger level
[TIP866F_NO | TIP866F_T8 | TIP866F_T16 | TIP866F_T32 | TIP866F_T56]

RR the least significant byte selects the receive trigger level
[TIP866F_NO | TIP866F_R8 | TIP866F_R16 | TIP866F_R56 | TIP866F_R60]

Disabling the FIFO is only possible if both FIFO triggers are set to TIP866F_NO. If not both FIFO triggers are set to TIP866_NO, the second one will be automatically set to TIP866F_x8.

The FIFO trigger-levels may influence the behavior of the target system. A modification of the FIFO-trigger-levels also means changing the duration of a single interrupt and the number of interrupts that will be generated.

Increasing the receive FIFO-trigger-level will lower the number of generated interrupts, but it will also increase the execution time of a single interrupt function and it may increase the risk of losing data by FIFO overrun.

Increasing the transmit FIFO-trigger-level will increase the number of generated interrupts, but it will also lower the execution time of a single interrupt function and decrease the chance of gaps in the transmission stream.

EXAMPLE

```
#include “tip866.h”

int             fd;
int             retval;

…
```

…

```
/*-----------------------------------------------
  Execute ioctl() function, set receive trigger level
  to 56 and transmit trigger level to 8
  -----------------------------------------------*/
retval = ioctl(fd,
               FIO_TIP866_FIFO,
               ((TIP866F_T8 << 8) | TIP866F_R56));
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual) or a driver set code described below. Function specific error codes will be described with the function.

Error Code	Description
S_tip866Drv_IARG	Invalid FIFO trigger value.

4.5.3 FIO_TIP866_DATABITS

This I/O control function sets the number of data bits for serial communication. The function specific control parameter **arg** specifies the new configuration

arg

This parameter selects the number of data bits in one word. The argument can be set to [TIP866DB_5 | TIP866DB_6 | TIP866DB_7 | TIP866DB_8] for 5 to 8 data bits.

EXAMPLE

```
#include “tip866.h”

int           fd;
int           retval;

/*-----------------------------------------------------
  Execute ioctl() function, set 7 Bit data word length
  -----------------------------------------------------*/
retval = ioctl(fd, FIO_TIP866_DATABITS, TIP866DB_7);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual) or a driver set code described below. Function specific error codes will be described with the function.

Error Code	Description
S_tip866Drv_IARG	Invalid data word length. Check ioctl parameter arg.

4.5.4 FIO_TIP866_STOPBITS

This I/O control function sets the number of stop bits. The function specific control parameter **arg** specifies the new configuration.

arg

This parameter selects the size of the stop bit(s). Allowed values are *TIP866SB_10* for one, *TIP866SB_15* for 1.5 and *TIP866SB_20* for 2 stop bits.

EXAMPLE

```
#include “tip866.h”

int           fd;
int           retval;

/*----------------------------------------
  Execute ioctl() function, use one stop bit
  ----------------------------------------*/
retval = ioctl(fd, FIO_TIP866_STOPBITS, TIP866SB_10);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual) or a driver set code described below. Function specific error codes will be described with the function.

Error Code	Description
S_tip866Drv_IARG	Invalid stop bits length. Check ioctl parameter arg.

4.5.5 FIO_TIP866_PARITY

This I/O control function sets the parity checking parameters. The function specific control parameter **arg** specifies the new configuration.

arg

This parameter selects parity checking. Parity checking can be set to even (*TIP866EVP*) or odd (*TIP866ODP*) parity, or it can be disabled (*TIP866NOP*).

EXAMPLE

```
#include “tip866.h”

int           fd;
int           retval;

/*---------------------------------------
  Execute ioctl() function, use odd parity
  ---------------------------------------*/
retval = ioctl(fd, FIO_TIP866_PARITY, TIP866ODP);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual) or a driver set code described below. Function specific error codes will be described with the function.

Error Code	Description
S_tip866Drv_IARG	Invalid parity checking parameter. Verify ioctl parameter arg.

4.5.6 FIO_TIP866_ENABLEHWHS

This I/O control function enable hardware handshake for certain channel. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include “tip866.h”

int           fd;
int           retval;

/*-----------------------------------------------------
  Execute ioctl() function, enable hardware handshake
  -----------------------------------------------------*/
retval = ioctl(fd, FIO_TIP866_ENABLEHWHS, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

4.5.7 FIO_TIP866_DISABLEHWHS

This I/O control function disables the hardware handshake for the certain channel. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include “tip866.h”

int           fd;
int           retval;

/*-----------------------------------------------------
  Execute ioctl() function, disable hardware handshake
  -----------------------------------------------------*/
retval = ioctl(fd, FIO_TIP866_DISABLEHWHS, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

4.5.8 FIO_TIP866_CHECKBREAK

This I/O control function looks for break conditions. The function specific control parameter **arg** is a pointer to result buffer.

This function checks, if break has been received since device creation, reconfiguration or the last *FIO_TIP866_CHECKBREAK* call.

arg

This parameter is a pointer to a char value, where the result will be stored to. A result of TRUE means that a break has been received. A result of FALSE means that no break has been received. The input break condition will be deleted with this call.

EXAMPLE

```
#include “tip866.h”

int           fd;
char          breakCheck;
int           retval;

/*-----------------------------------------------
  Execute ioctl() function, check break condition
  -----------------------------------------------*/
retval = ioctl(fd, FIO_TIP866_CHECKBREAK, (TIP866_IOCTL_ARG_T)&breakCheck);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

4.5.9 FIO_TIP866_SETBREAK

This I/O control function sets the break bit of the controller. This will produce a break signal on the transmit line. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include “tip866.h”

int           fd;
int           retval;

/*-------------------------------------------------------------------
  Execute ioctl() function, create break condition on transmit line
  -------------------------------------------------------------------*/
retval = ioctl(fd, FIO_TIP866_SETBREAK, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

4.5.10 FIO_TIP866_CLEARBREAK

This I/O control function removes the break flag of the controller. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include “tip866.h”

int           fd;
int           retval;

/*-------------------------------------------------------------------
  Execute ioctl() function, clear break condition on transmit line
  -------------------------------------------------------------------*/
retval = ioctl(fd, FIO_TIP866_CLEARBREAK, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

4.5.11 FIO_TIP866_RECONFIGURE

This I/O control function reconfigures the selected channel. The driver internal settings will be set to the default configuration and the channel will be set up with its default settings. The function specific control parameter **arg** is not used for this function.

EXAMPLE

```
#include “tip866.h”

int           fd;
int           retval;

/*-------------------------------------------------------------
  Execute ioctl() function, load and set default configuration
  -------------------------------------------------------------*/
retval = ioctl(fd, FIO_TIP866_RECONFIGURE, 0);
if (retval != ERROR)
{
     /* function succeeded */
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).

4.5.12 FIO_TIP866_CHECKERRORS

This I/O control function checks, if errors were detected since device creation, reconfiguration or the last *FIO_TIP866_CHECKERRORS* call. This call needs the pointer to a char value, where the result will be returned. The result is a flag field with bits set for an error condition. The function specific control parameter **arg** is a pointer to a char value, where the result will be returned.

arg

Result pointer. The following flags are valid:

Bit	Error
TIP866_FRAMING_ERR	Framing Error
TIP866_PARITY_ERR	Parity Error

EXAMPLE

```
#include “tip866.h”

int           fd;
char          errorCheck;
int           retval;

/*-----------------------------------------------
  Execute ioctl() function, check receive errors
  -----------------------------------------------*/
retval = ioctl(fd,
               FIO_TIP866_CHECKERRORS, (TIP866_IOCTL_ARG_T)&errorCheck);
if (retval != ERROR)
{
     /* function succeeded */
     if (errorCheck & TIP866_FRAMING_ERR)
     {
          /* handle framing errors */
     }

     if (errorCheck & TIP866_PARITY_ERR)
     {
          /* handle parity errors */
     }
}
else
{
     /* handle the error */
}
```

RETURN VALUE

OK if function succeeds or ERROR.

ERROR CODES

The error codes are stored in *errno* and can be read with the function *errnoGet()*.

The error code is a standard error code set by the I/O system (see VxWorks Reference Manual).